Entry 4: Potion game jam

Project discussion

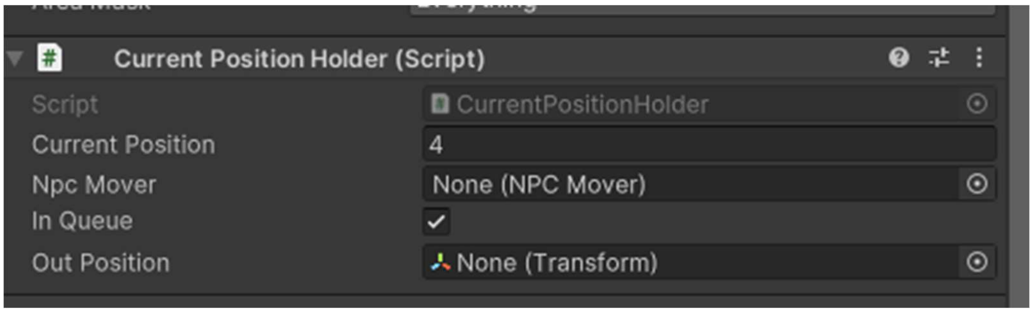
This project was the final game jam I completed in my first year, and was one of my favourites out of all, as it combined the techniques I had learned in previous game jams to produce something that I am very proud of.

This project is a potion brewing simulator, similar to “Overcooked”, and a collaboration between two designers, two artists, myself and one other programmer.

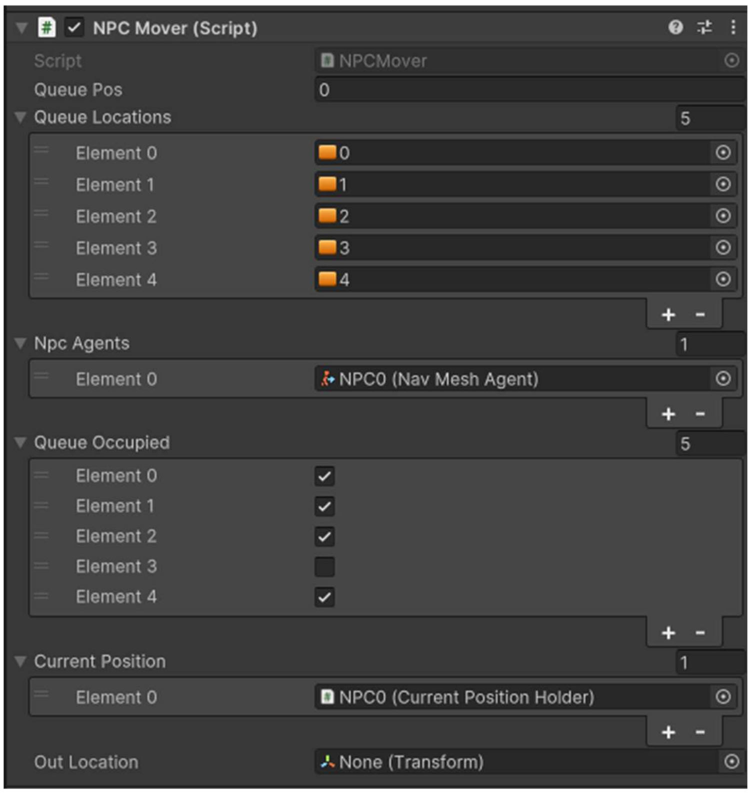
Actions and decisions

During this project, I was responsible for the movement, queues, instantiation of NPCs and a dynamic system that generates potion orders and stores displays the progress of such on the UI.

The NPC queue system was something that took a while to figure out and was eventually scrapped and re-made. The first system I had tried to implement, was not entirely thought out, and I did not attempt to utilise classes, instead opting for a system of separate arrays and references across multiple different scripts to move my NPCs.


Current Position Holder (Script)

Script	CurrentPositionHolder
Current Position	4
Npc Mover	None (NPC Mover)
In Queue	✓
Out Position	None (Transform)


NPC Mover (Script)

Script	NPCMover
Queue Pos	0
Queue Locations	5
Element 0	0
Element 1	1
Element 2	2
Element 3	3
Element 4	4
Npc Agents	1
Element 0	NPC0 (Nav Mesh Agent)
Queue Occupied	5
Element 0	✓
Element 1	✓
Element 2	✓
Element 3	
Element 4	✓
Current Position	1
Element 0	NPC0 (Current Position Holder)
Out Location	None (Transform)

My NPCs at this stage are capsule objects, that contain a nav mesh agent in a nav mesh area. Programming at this stage, I did not consider the future requirements for each NPC to store its own requested orders and accommodate for other features we had planned.

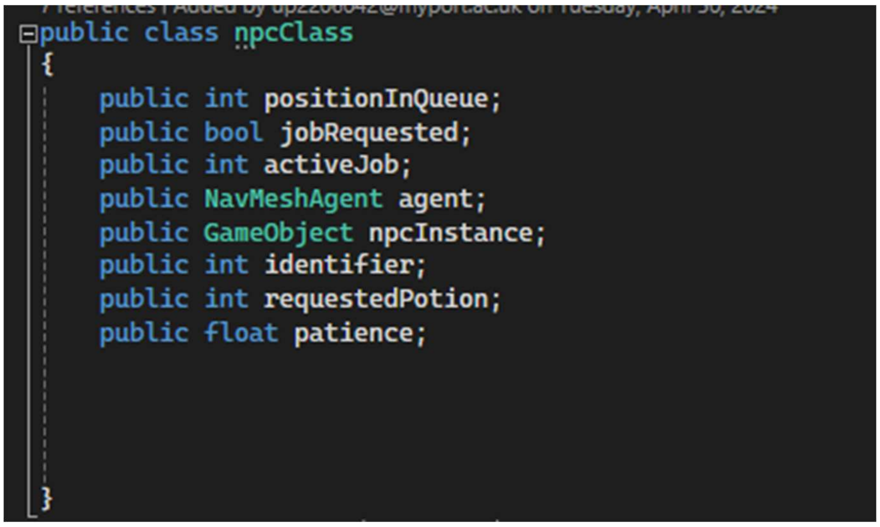
The array system worked when I tried to move NPCs and each NPC that I created would follow the queue as intended. However, issues with this approach would present themselves very quickly.

The first issue I encountered was the fact that due to my use of arrays, the number of NPCs that could be spawned in at once was not easily scalable.

The second issue was that when I tried to combine my system with my teammate's code, requiring each NPC to store an identifier for its own requested potion, my system of arrays could not accommodate for that.

Eventually, I scrapped my entire queue system in favour of something more dynamic.

Looking back on my previous projects and workshops, I decided to re-implement my NPC system using classes. Combining this with the replacement of arrays in favour of lists, allowed me to create a reference list of all NPCs in game, which automatically completed some of the functionality I needed in my previous system.

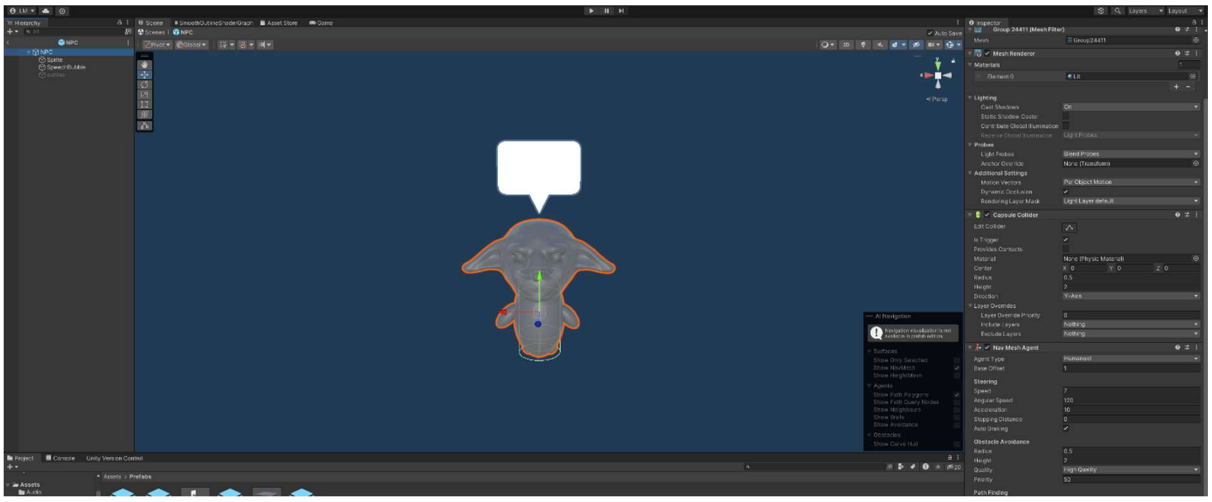


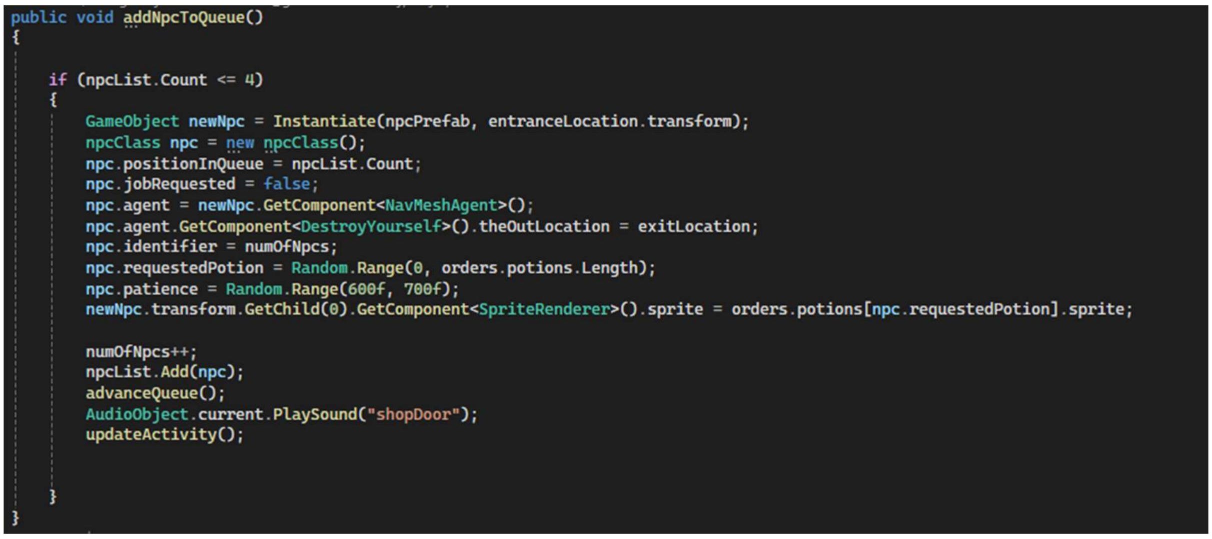
```
public class npcClass
{
    public int positionInQueue;
    public bool jobRequested;
    public int activeJob;
    public NavMeshAgent agent;
    public GameObject npcInstance;
    public int identifier;
    public int requestedPotion;
    public float patience;




}
```

Due to the addition of classes, whenever I wanted to create a new NPC in the scene, I would instantiate a new NPC game object from a prefab





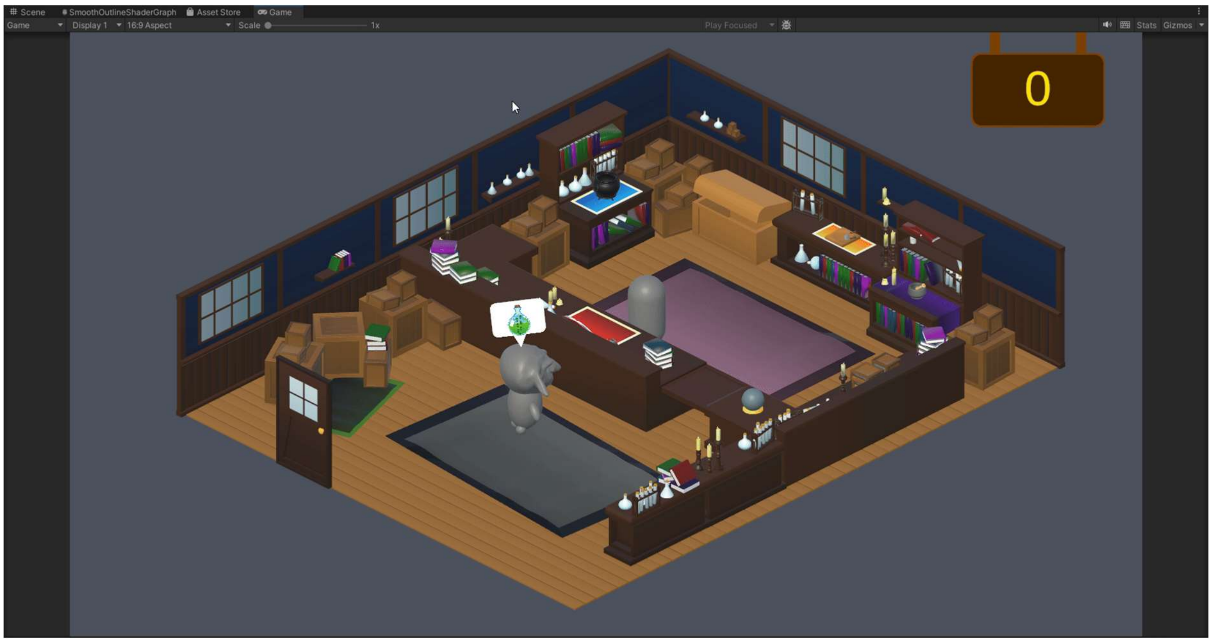
```
public void addNpcToQueue()
{

    if (npcList.Count <= 4)
    {
        GameObject newNpc = Instantiate(npcPrefab, entranceLocation.transform);
        npcClass npc = new npcClass();
        npc.positionInQueue = npcList.Count;
        npc.jobRequested = false;
        npc.agent = newNpc.GetComponent<NavMeshAgent>();
        npc.agent.GetComponent<DestroyYourself>().theOutLocation = exitLocation;
        npc.identifier = numOfNpcs;
        npc.requestedPotion = Random.Range(0, orders.potions.Length);
        npc.patience = Random.Range(600f, 700f);
        newNpc.transform.GetChild(0).GetComponent<SpriteRenderer>().sprite = orders.potions[npc.requestedPotion].sprite;

        numOfNpcs++;
        npcList.Add(npc);
        advanceQueue();
        AudioObject.current.PlaySound("shopDoor");
        updateActivity();


    }
}
```

This approach ended up working much more effectively, allowing me to spawn in NPCs on a timer, since everything was handled mostly within one function. This also allowed my teammate to implement their systems without the need to navigate several connected scripts.



Since this approach worked well, I re-used and adapted this for my “orders request queue”.

Instead of having to tell each order where it is placed on screen, that functionality was completed automatically by utilising a horizontal box in the UI.

Each order element is its own prefab, which is instantiated into the horizontal box when the order is added. It contains its own code for the timer bar and self-deletion on expiry.

Reflections

I know the use of classes and instantiation was the best approach in these scenarios, as it demonstrated its own effectiveness, when we went to implement and expand our systems.

Being able to avoid the sunk cost fallacy paid off once again, as I saved more time by re-making the system rather than trying to make it adapt to everything I would add on top.

Insight

Being comfortable in using classes is something I will continue to practice, as it has proven to be a highly diverse and effective tool. Furthermore, constructing my functions so that another programmer can implement their own code saved a lot of time and effort, allowing us to evenly divide our workloads.

What I would do differently

There is not much I would do differently, aside from planning ahead to organise progress and time better, as we were cut off in development by the deadlines, leaving the game feeling incomplete.